Anweisung	Fujitsu/LSI #1,#2	Cypress/ Fujitsu #3	MicroSPARC (&B5000)	Fujitsu SPARClite	Super SPARC
# of register windows	7	8	7	8	8
ld (32 bit integer)	2	2	1	1	1
ldd (64 bit integer)	3	3	2	2	1
ld (32 bit float)	2	2	1	1	1
ldd (64 bit double)	3	3	2(1)	2	1
st (32 bit integer)	3	3	2	1	1
std (64 bit integer)	4	4	3	2	1
st (32 bit float)	3	3	2(1)	1	1
std (64 bit double)	4	4	3(2)	2	1
taken branches	1	1	1	1	1
untaken branches	2	1	1	1	1
jmpl and rett	2	2	2	2	1
integer multiply	N/A	N/A	19	?	4
integer divide	N/A	N/A	39	?	18
issue FP operation	2	1	1	N/A	1

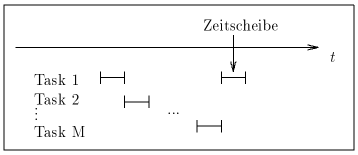
Tabelle 1.23: Anzahl der Register-Fenster und Integer-Zyklen pro Anweisung

Ein Ausflug in Multiprozess- und Multiprozessorsysteme

Prozesse (Tasks)

- Programmeinheiten, die unterbrech- und wiederfortsetzbar sind
- Multitasking:

  Ein Rechner erscheint M Benutzern als M jeweils dedizierte einzelne Rechner. Ein Rechner kann (scheinbar) zugleich mehrere Prozesse bearbeiten.



Beispiele:

Windows: Multitasking, nicht preemptiv

Windows NT / UNIX: preemptives Multitasking

Anweisung	FPC & TI 8847	Weitek 3170 & 3171	Texas 602	BIT B5000	MicroSPARC min	MicroSPARC typ	MicroSPARC max	Super SPARC
fitos	8	10	4	2	5	6	13	1
fitod	8	5	4	2	4	6	13	1
fstoir, fstoi	8	5	4	2	6	6	13	1
fdtoir, fdtoi	8	5	4	2	7	7	14	1
fstod	8	5	4	2	2	2	14	1
fdtos	8	5	4	2	3	3	16	1
fmovs	8	3	4	2	2	2	2	1
fnegs	8	3	4	2	2	2	2	1
fabss	8	3	4	2	2	2	2	1
fsqrts	15	60	22	24	6	37	51	6
fsqrtd	22	118	32	45	6	65	80	10
fadds, fsubs	8	5	4	2	4	4	17	1
faddd, fsubd	8	5	4	2	4	4	17	1
fmuls	8	5	4	3	5	5	25	1
fmuld	9	8	6	4	7	9	32	1
fdvis	13	38	16	14	6	20	38	4
fdivd	18	66	26	24	6	35	56	7
fcmps, fcmpes	8	3	4	2	4	4	15	1
fcmpd, fcmped	8	3	4	2	4	4	15	1

Tabelle 1.24: „floating point"-Zyklen pro Anweisung

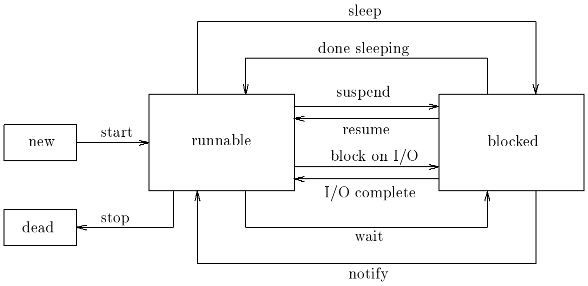
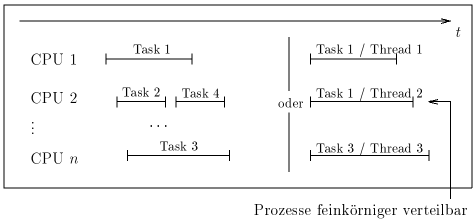


Abbildung 1.24: Thread/Prozeß-Stati

Preemtiv (=*wahlweise vergebbar*) heißt ein Multitasking-Betriebssystem dann, wenn der Betriebssystemkern allein für die Vergabe der Zeitscheiben verantwortlich ist, also nicht auf die kooperative Mitarbeit der Tasks angewiesen ist. Die Tasks werden von *außen* unterbrochen und später wieder fortgesetzt.

Threads – parallele, unabhängig durchführbare Teile eines Prozesses, die deshalb auf Multiprozessorsystemen ($\neq$ Multiprozesssystemen) durch verschiedene CPU's durchgeführt werden können: An der Aufgabe eines Prozesses arbeiten mehrere Prozessoren zugleich.



Wegen des gemeinsam benutzten Speichers ist ein gegenseitiger Ausschluß auf den Zugriff nötig:


```
⋮
while (Guthaben<Auszahlungsbetrag) do
   wait();
Guthaben := Guthaben - Auszahlungsbetrag;
Auszahle(Auszahlungsbetrag);
notify();
⋮
```

Threads sind weniger resourcenintensiv als die Erzeugung weiterer Prozesse. Mehr Probleme lassen sich in Threads zerlegen als in mehrere fast unabhängige Prozesse. Dafür sind Thread-Algorithmen schwieriger zu programmieren.

1.2.2.7 Einfügen und Löschen in verketteten Listen

Verkettete Listen (*linked list*) sind dynamische Datenstrukturen, die von multithreaded Anwengungen genutzt werden können. Ein Lock [3] kann dazu benutzt werden, den Zugriff auf die Daten zu synchronisieren. Einfache Locks zum Schutz von großen und häufig von mehreren Threads gemeinsam benutzten Datenstrukturen können jedoch uneffizient sein. Im folgenden Beispiel wird die CAS Anweisung dazu gebraucht, um auf einer verketteten Liste ohne explizite Locks zu operieren. Jedes Listenelement beginnt im Beispiel mit einem Adressenfeld, das entweder die Adresse des nächsten Listenelements enthält oder leer ist. Im Listenkopf ist die Adresse einer Variablen gespeichert, die die Adresse des ersten Listenelements enthält. Bei leeren Listen steht im Kopf eine Null.

```
ListInsert(Head Element) !%i0 = Head addr, %i1 = Element addr
retry:
  ld    [%i0],%l0
  st    %l0, [%i1]
  mov   %i1, %l1
  cas   [%i0],%l0,%l1
  cmp   %l0,%l1
  bne   retry
  nop
ListRemove(Head) !%i0 = Head addr
retry:
  ld    [%i0],%o0
  tst   %o0
```

[3] Lock=Sperrvorrichtung, sorgt dafür, daß einer Resource genau einem Thread zugeteilt wird, bis dieser sie wieder freigibt (gegenseitiger Ausschluß).